Sink Controlled Reliable Transport for Disaster Recovery

Charilaos Stais, George Xylomenos, Giannis F. Marias
Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business,
Athens, Greece
Email:{stais, xgeorge, marias}@aueb.gr

***Abstract*—We present a reliable transport layer protocol for sensor networks, targeting disaster recovery applications where human or robotic rescuers try to gather information from a possibly fragmented sensor network by moving through the disaster area. The mobility of the information sink means that the protocol must quickly adapt to a constantly changing view of the network, where connections and disconnections are the norm. Our protocol is purely sink driven, that is, the sink controls congestion by rate limiting the sensors, choosing how to assign the available bandwidth to different sensor types and deciding on the level of reliability to be achieved. In addition, our protocol operates at the application layer with minimal requirements from lower layers, allowing its integration with a disaster recovery application that will set its parameters depending on the disaster scenario. As a result, our protocol allows simple and inexpensive fixed sensors to be combined with expensive but reusable mobile equipment for disaster recovery purposes.**

***Index Terms*—Sensor networks, reliable transport, congestion control.**

I. Introduction

One possible use of sensor networks is assisting robotic or human rescuers in disaster recovery. In such scenarios, a mobile rescuer roams a disaster area, for example, a building hit by a fire or an earthquake, gathering information from any available sensors. For example, temperature sensors can indicate whether nearby areas are on fire, chemical sensors can detect the presence of people breathing, while audio and video sensors can reveal what is behind a blocked passage. The *DIstributed Sensor systems For Emergency Response* (DISFER) project [1], aims to advance the state of the art in disaster recovery via sensor network technologies.

In such scenarios, the sensor network may only be partially connected, as the disaster may have wiped out parts of the infrastructure. However, as the rescuer moves, it will come into contact with most sensors, either directly, or via other sensors in a multihop configuration. As a result, the transport protocol used to move data from the sources (sensors) to the sink (rescuer) must establish connections and exchange information very quickly, as the sink moves within range of the sensors, reconfiguring itself as the rescuer moves around. In addition, the transport protocol must be mostly reliable, since the rescuer cannot count on sensor redundancy to reveal critical information. Finally, as data from many sources converge towards the sink, the transport protocol must avoid losing packets due to congestion around the sink.

It is crucial to note that all these requirements revolve around the sink: the sink must receive data with a high probability of success, sink mobility makes the network change as nodes come in and out of the network, and it is the area near the sink that is most likely to be congested. For this reason, in this paper we present a reliable transport protocol for sensor networks with mobile sinks that is purely sink-controlled, in the sense that the sink allocates transmission rates to all reachable sources and manages the error recovery process, depending on application objectives.

By moving all the intelligence to the sink, we get many benefits. First, the sources are simpler, hence cheaper and with longer battery lives; the sink only needs to operate for a short period of time, therefore it can spend more energy, and it can be reused to amortize its cost. Second, the sink can adapt its behavior to the problem at hand; for example, if congestion arises it can allocate higher transmission rates to the sensors that are of higher importance to a specific mission. Third, the sink-rescuer is the only mobile part, therefore it can adapt its mobility pattern to the data coming in from the sensors; for example, it can stay longer at some place in order to gather more data, or move to a different spot for better reception. Finally, by properly designing the error control mechanism, the sink can also set the required level of reliability, depending on the type and importance of each sensor.

The remainder of this paper is organized as follows. In Section II we present our assumptions and motivate our design choices. In Section III we describe how our protocol operates and in Section IV we explain the rate control scheme used to handle congestion. Section V describes our prototype implementation. In Section VI we discuss related work and present our conclusions in Section VII.

II. Assumptions and Rationale

We assume a set of sensors operating as a multihop network, using a shared channel (e.g. WiFi or Bluetooth) for communication. Each sensor acts as a source attempting to transmit data to a mobile sink (a robotic or a human rescuer equipped with a computer). We assume that all sensors are

[1] http://www.aueb.gr/disfer

preprogrammed with the network address of the sink. The network may become disconnected due to sensor failures, therefore, as the sink moves it may only be able to reach a subset of the sensors. Whenever we refer below to *the network*, we mean the sink and the sources that are reachable by the sink via one or more hops at any given time. As the sink moves, this subset changes. We further assume an underlying routing protocol that attempts to route data between all the nodes in the currrent network. We do not enforce a specific protocol, but note that a routing protocol based on the *received signal strength* (RSSI) metric, available in most wireless interfaces, which triggers route recalculations whenever the sink detects that the node with the highest RSSI has changed [1], is suitable for our approach.

The sensors are distinguished in multiple types, with each type being separately rate-controlled. The sink dedicates a portion of its available bandwidth to each sensor type, and then assigns part of that bandwidth to each individual sensor of that type. While for our protocol all sensors are basically data sources that require a specific transport rate and may alternate between active and idle states, the distinction of sensors into multiple types allows applications to decide how to partition the available bandwidth between the types and, possibly, operate different bandwidth allocation schemes for each type. To illustrate the sensor type concept, in our prototype we assume two types, event and continuous sensors. An *event sensor* collects data (e.g. a temperature value or a camera snapshot) on a periodic basis, while a *continuous sensor* sends a continuous stream of data (e.g. live video or audio). In our setting, event sensors send data periodically and not only when an event occurs (e.g. when the temperature changes or when movement is detected), since the mobile sink will need to gather as much data as possible when the sensor is reachable, which may only be a brief period.

In terms of congestion control, in our protocol the sink *explicitly* controls the transmission rates of all sources. Since all source transmissions converge at the sink, the sink is at the best position to detect the onset of congestion and take measures to control it. Furthermore, the sink is aware of application requirements, i.e. which sensors are considered more important, so as to appropriately regulate their rates; this depends on the scenario and cannot be preprogrammed at the sensors. Finally, sink mobility means that the network topology changes all the time, thus complicating distributed congestion control. While sensors join and leave the network, the sink always remains in the network, hence it is the best place to implement congestion control. As a side effect, this makes sensors simpler, cheaper and with longer battery lives. Since our protocol targets mobile sinks, it implements a simple congestion control loop that allows the sink to quickly regulate each reachable source.

In terms of reliability, our protocol uses *negative acknowledgments* (NACKs) to trigger retransmissions of lost data but, unlike most protocols which immediately retransmit lost data, we transmit data in rounds: first all data packets are transmitted, then all NACKed packets are transmitted, then all NACKed retransmissions are transmitted again, and so on. This allows the sink to stop the recovery process whenever it deems appropriate, for example, when enough packets have been received to allow reconstructing the content. When the source loses connectivity with the sink before the transmission completes, the sink will have received packets from the entire transmission rather than only from the beginning, which may also be useful for approximately reconstructing the data.

Since rate allocation only operates at the sink, the sink can implement any rate allocation policy desired. In our prototype we explicitly split the sensors into classes depending on their type, performing separate rate allocation in each class. In this manner, event sensors always have a guaranteed chance to transmit data, while the remaining bandwidth can be used by the continuous sensors. The rationale is that while higher transmission rates for continuous sensors make them more useful (e.g. higher frame rates and resolutions lead to more informative video), they should be regulated to allow the low bandwidth event sensors to always transmit there data. The exact way the bandwidth is split is up to the application; it can be tuned to a specific rescue mission, taking into account the sensors in a disaster area. For example, in a building with a few sensors and many cameras, more bandwidth could be dedicated to continuous sensors before the rescuer enters the building.

A final aspect of our protocol is that it operates at the application level, using UDP/IP messages as the underlying transport. It can be implemented on any device offering IP connectivity and a UDP socket interface. Our prototype is written in Java, allowing it to run on, among other devices, Android smartphones and tablets, without kernel modifications or root priviledges. In addition to portability and ease of debugging, our implementation can be tightly integrated with the application using it, allowing the application to directly control aspects of protocol operation such as the allocation of rates to sensor classes and the level of reliability required.

III. Protocol Description

Communication between the sensors and the sink in our protocol proceeds in five stages: connection establishment, sensor information exchange, data exchange, idle and connection release.

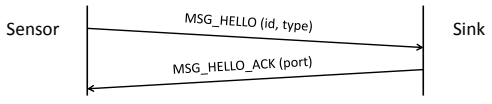

Fig. 1. Connection establishment stage.

A. Connection establishment

When the sink begins operation, it listens to a well-known UDP port for connection requests from sensors. The sensors wait until the sink becomes reachable before connecting to it. In our prototype, the sink periodically sends a probe message, `MSG_HELLO`, to the well-known IP address and UDP port of the sink, until it receives a response; the probe interval is configurable. If the routing protocol supports it, the sink

can ask the routing engine to be notified when the sink becomes reachable. The MSG_HELLO message includes the sensor identifier, unique in the sensor network, and its type; in our prototype, only event and continuous sensor types exist. When the sink receives such a message, it responds with an MSG_HELLO_ACK message which indicates a separate UDP port where the sensor should send the following control messages. The sink notes the time when it sent the message, so as to later measure the *round-trip time* (RTT) to that sensor. The use of a separate control port per sensor allows the sink to dedicate one thread to receiving new connection requests, and a separate thread for each connection to a specific sensor, thus avoiding the need to multiplex messages from multiple sensors over a single control connection. The connection establishment messages are depicted in Figure 1.

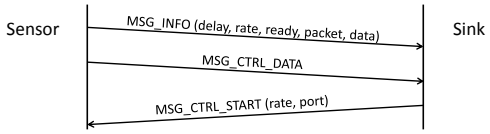

Fig. 2. Sensor information exchange stage.

B. Sensor information and idle

After receiving a response from the sink, the sensor prepares an MSG_INFO message which includes the delay between receiving the message from the sink and sending the response, the data rate requested by the sensor, whether the sensor is ready to send data at this point in time or not, the data packet size and the total size of the data to send. When the sink receives the MSG_INFO message it calculates the time elapsed since sending the MSG_HELLO_ACK, subtracting the delay in the message to get the RTT to the sensor. At this point, the connection has been established and the sink is aware of the sensor's bandwidth requirements. If the sensor has indicated that it is not ready to send data at this point, the sink moves to an idle state, waiting until the sensor sends a MSG_CTRL_DATA with no parameters. Then, the sink starts the data exchange by sending an MSG_CTRL_START message to the sensor, indicating the data rate to use and a UDP data port to use for the transmission. If the sensor indicated in the MSG_INFO message that it was ready to send data, the MSG_CTRL_START message is sent immediately as a response. The initial rate allocated to the sink is set as explained in Section IV. The sensor information messages are shown Figure 2.

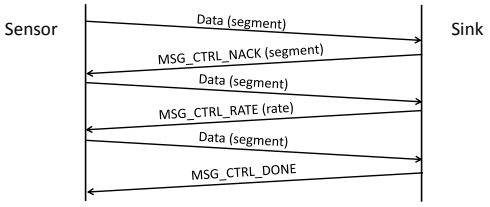

Fig. 3. Data exchange stage.

C. Data exchange

After the sink sends the MSG_CTRL_START, the actual data transfer begins, using the UDP data port assigned for the transfer; control messages, such as NACKs and rate updates, are exchanged out of band over the control channel, without being rate controlled. This allows control messages to be sent without waiting behind a, possibly long, queue of data messages. The sensor breaks down its transmission into packets with the size indicated in the MSG_INFO message, until all the data indicated in the MSG_INFO message is exhausted. Data packets only have a single header field, a segment number used to sequentially number all data packets.

When a missing packet is detected, the sink sends a MSG_NACK to the sensor over the control channel. However, the sensor does not immediately retransmit lost messages. After the transmission is complete, all missing messages are retransmitted, generating further NACKs from the sink, if needed. This procedure is repeated in rounds, until all messages are received [2]. Once recovery is complete, the sink sends a MSG_CTRL_DONE message to indicate a successfully completed data transfer. Both endpoints then move to an idle state, until the sensor generates a new MSG_CTRL_DATA message. If the need arises, the sink will send a MSG_CTRL_RATE message to the sensor indicating its new rate allocation, as explained in Section IV. The data messages are shown in Figure 3.

Note that round-based recovery allows the sink to use any packets received without waiting for retransmissions. The sink may even stop the recovery process by sending the MSG_CTRL_DONE message. For example, when an image is transmitted using redundancy coding, the sink may stop the recovery process when enough packets have been received to adequately recostruct the image. This allows the application to fine tune the reliability of the protocol.

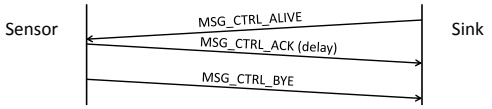

Fig. 4. Connection control and release stage.

D. Connection control and release

Since the path between the sink and the sensor may become disconnected due to the sink's mobility, the connection may fail between data transfers, without either side noticing. For this reason, the sink periodically sends an MSG_CTRL_ALIVE message to the sensor, which is acknowledged by an MSG_CTRL_ACK message from the sensor that includes the delay incurred between receiving the MSG_CTRL_ALIVE and responding with the MSG_CTRL_ACK. In addition to confirming that the connection is still alive, this procedure allows the sink to periodically measure the RTT of the connection. If the sensor or the sink wishes to complete the connection, they can send a MSG_CTRL_BYE message, which does not need to be acknowledged, as after either side drops the connection,

the other one will eventually timeout: the sink times out if no responses are received to its MSG_CTRL_ALIVE messages, while the sensor times out if no MSG_CTRL_ALIVE messages arrive. The MSG_CTRL_BYE message simply allows the other end to release the resources dedicated to the connection without waiting for a timeout. The connection control messages are shown in Figure 4.

IV. Congestion management

The congestion management mechanism of our protocol is agile and purely sink-driven. Since we focus on congestion around the sink, we know the total available bandwidth as it depends on the technology used by the sink and sensors for data exchange. We first reserve a fixed part of this bandwidth, e.g. 30%, for the control message exchanges which are *not* rate-controlled. Then, the sink splits the remainder to event and continuous sensors using a ratio determined by the application, e.g. 10%-90%, depending on the number and type of sensors present at the disaster site.

The congestion management algorithm periodically evaluates the state of individual connections and the system as a total. The sink monitors the RTT of each connection using the MSG_HELLO_ACK, MSG_INFO, MSG_CTRL_ALIVE and MSG_CTRL_ACK messages; the processing delay at the sensor is always subtracted to get an accurate RTT estimate. The congestion management algorithm maintains the last few RTT samples and their moving average.

Whenever the algorithm runs, it first checks whether the average for each sensor has increased compared to the previous value by more than a configurable threshold. If this occurs four times in a row, then the corresponding connection is congested, otherwise it is not. If the connection is congested, the sink instructs the sensor to reduce its rate by 20%, via a MSG_CTRL_RATE message.

After each individual sensor is checked, if a new sensor has been connected or an existing one has been disconnected, the entire system is checked to see whether global adjustments need to be made. This takes places separately for each sensor class. First, we calculate the total rates requested (not assigned) by the sensors of the class. If these are below the available bandwidth, they will all get what they asked for. New sensors will get their requested rate in the MSG_CTRL_START message which directs them to start sending data. Sensors that were previously rate limited, will increase their rate by 20%, while other sensors will get their requested rate; in both cases, the change is announced via a MSG_CTRL_RATE. If, on the other hand, the requested badnwidth is higher than the available one, the available rate is shared *equally* among all sensors of that class. The sensors are notified as above, i.e. either via a MSG_CTRL_START or via a MSG_CTRL_RATE message.

The congestion management algorithm is very simple, as we expect congestion to be concentrated around the sink. Since the sink is constantly on the move, it is very unlikely that a distributed congestion control management will have time to converge. While TCP uses an *Additive Increase - Multiplicative Decrease* (AIMD) algorithm, our scheme uses fixed and symmetric steps. This is because TCP sources constantly probe the network for capacity, hence entering deep into the congested region before having to abruptly backoff. In our scheme sensors are conservatively rate controlled, hence congestion is expected to appear slowly, therefore there is no need for dramatic rate reductions.

V. Implementation

Our prototype was implemented in Java, consisting of around 20 KB of bytecode for the sensor and 40 KB of bytecode for the sink. The implementation runs entirely at the user level, allowing it to be compiled jointly with the application using it. The prototype uses configuration files to set the behavior of the protocol, for example, the shares of bandwidth between the sensor categories; these can instead be set directly by the application. Similarly, the prototype uses files stored on disk in lieu of actual sensor data; these can instead be objects generated by the sensor. A well-known UDP port and IP address needs to be agreed between the sink and sensors to allow them to rendezvous, but the additional control and data ports for each sensor are assigned automatically by the protocol. The protocol does not require any changes to the kernel or the libraries of the operating system, or even superuser access, since it operates over simple UDP/IP sockets.

VI. Related Work

There is a large body of work on transport protocols for wireless sensor networks. According to the taxonomy in [3], we can classify transport protocols in two axes, depending on their approach to reliability and congestion control. In terms of reliability, a protocol may offer unreliable or reliable service, while in terms of congestion control, a protocol may offer no congestion control, distributed congestion control or centralized congestion control. Actually, reliability can be further subdivided: hop-by-hop reliability via retransmissions, as in RMST [4], end-to-end reliability via retransmissions, as in RCPT [3] and STCP [5], and forward reliability without retransmissions, as in ReInForM [6]. Our protocol implements reliable service with centralized congestion control, with reliability achieved an end-to-end via retransmissions.

While the hop-by-hop reliability of RMST is useful for a wireless environment, most wireless networks can retransmit lost packets at the link layer, an approach that works well enough [4]. We therefore concentrate on congestion induced losses, as in RCPT and STCP, which we handle end-to-end. The use of multiple transmissions without NACKs, as in ReInForM, requires a well-connected sensor network, which is unlikely in disaster recovery applications. For congestion control, we chose a centralized approach as in RCPT for many reasons. First, as all data converge at the sink, packet drops will occur if the sensors are not somehow regulated. Second, in our target application the sink is mobile, therefore distributed congestion control would probably never converge. Third, by concentrating all congestion control decisions at the sink as in [3], we can modify its behavior depending on the environment and ap;lication. Fourth, this approach does not

require sensors to implement any congestion control measures as in [5].

Our protocol is most similar to RCPT [3], as it is based on the sink explicitly controling the transmission rates of the sensors and sending NACKs to the sensors. The differences between RCPT and our protocol are due to the fact that we explicitly address mobile sinks, as in COSMOS [1], therefore we have implemented simpler and faster control loops than RCPT, while avoiding the distributed congestion control of COSMOS which we expect to be slow to converge. On the other hand, while both STCP and ReInForM provide limited reliability, our approach allows the sink to dynamically define and control the reliability level depending on the application, unlike ReInForM where the reliability goal is fixed when a packet is generated and never changes [6], and STCP where the sink control reliability but the sensors set the reliability goals [5].

VII. Conclusion

We have presented a reliable transport protocol for sensor networks which is especially suitable for disaster recovery applications. Our protocol assumes that the sink is mobile, thus requiring a fast and agile method for congestion control. The protocol is purely sink driven, thus allowing application policies to be set without previously configuring the sources. Furthermore, it allows the available bandwidth to be split between different classes of sensors and within the sensors of each class depending on application preferences. The reliability level achieved can also be fine tuned by the application, which can be very tightly integrated with the protocol, as they both operate at the application level.

Acknowledgment

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS - Athens University of Economics and Business - DISFER.

References

[1] K. Karenos and V. Kalogeraki, "Traffic management in sensor networks with a mobile sink," *IEEE Trans. on Parallel and Distributed Systems*, vol. 21, no. 10, pp. 1515–1530, 2010.

[2] C. Stais, A. Voulimeneas, and G. Xylomenos, "Towards an error control scheme for a publish/subscribe network," in *Proc. of the IEEE ICC*, 2013, pp. 3743–3747.

[3] J. Paek and R. Govindan, "RCRT: Rate-controlled reliable transport for wireless sensor networks," in *Proc. of ACM SenSys*, 2007, pp. 305–319.

[4] F. Stann and J. Heidemann, "RMST: reliable data transport in sensor networks," in *Proc. of the IEEE SNPA Workshop*, 2003, pp. 102–112.

[5] Y. Iyer, S. Gandham, and S. Venkatesan, "STCP: a generic transport layer protocol for wireless sensor networks," in *Proc. of the ICCCN*, 2005, pp. 449–454.

[6] B. Deb, S. Bhatnagar, and B. Nath, "ReInForM: reliable information forwarding using multiple paths in sensor networks," in *Proc. of the IEEE LCN*, 2003, pp. 406–415.